Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Khanh Hoang Nguyen

End-to-end testing on Web Experience Management Platform

Master's Thesis
Espoo, Feb 29, 2020

Supervisor: Professor Petri Vuorimaa
Advisor: Markku Mantere M.Sc. (Tech.)

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Khanh Hoang Nguyen		
Title:	End-to-end testing on Web Experience Management Platform		
Date:	Feb 29, 2020	**Pages:**	vii + 56
Major:	Software and Service Engineering	**Code:**	SCI3043
Supervisor:	Professor Petri Vuorimaa		
Advisor:	Markku Mantere M.Sc. (Tech.)		

End-to-end testing (E2E) is the highest level of testing that verifies the entire system under test. Because tests of this kind are very expensive to develop and execute, they are often automated and used only to test critical functionalities of the system. Despite the abundance of non-academic resources, literature content on this topic is still lacking.

Web Experience Management System (WEMP) is a new solution for developing website functionality, where the consultancy company injecting their script directly into customer websites. While providing the flexibility, this also introduces new challenges since development work by customer will not be tested during consultancy workflow and vice-versa. Including E2E testing in the current process could help to guard against those misalignment.

The goal of this thesis is to implement an automated E2E testing into the current pipeline of an WEMP. The work consists of finding requirements for the needed artifacts and evaluating the result. The process is based on design science research where the artefact is the integration of E2E testing into the pipeline according to the requirements of a case client project. The result of the project is the automated E2E testing pipeline with integrated Continuous Integration (CI), Continuous Delivery (CD) and monitoring setup required for the case project.

The result conforms with the preliminary study on E2E testing. It showed how expensive and fragile E2E tests are. In conclusion, E2E tests are helpful but they should be limited to the most crucial customer journey. Moreover, tests should be implemented so that execution and reporting practices are useful for developers.

Keywords:	end-to-end testing, test automation, continuous integration, continuous delivery, web
Language:	English

Acknowledgements

I would especially want to thank my professor Petri Vuorimaa for his constructive suggestion and professional feedback throughout the project. I would also like to thank Juuso Vuoristo for his enthusiastic encouragement and valuable recommendation on the thesis. Additional thanks to Markku Mantere for making the case study project possible. Finally, I wish to thank my friends and family for their support during my study.

Espoo, Feb 29, 2020

Khanh Hoang Nguyen

Abbreviations and Acronyms

CI	Continuous Integration
CD	Continuous Deployment
E2E	End-to-end
FCP	Frosmo Control Panel
PM	Project Manager
QA	Quality Assurance
SUT	System Under Test
UI	User Interface
VCS	Version Control System
WEMP	Web Experience Management Platform

Contents

Chapter 1

Introduction

1.1 Motivation

There has been an enormous increase in the popularity of web application in the past decades. Websites and web-based solutions can be seen in every aspect of the our daily economic, social and educational activities. Nevertheless, assuring web application quality is usually the most challenging among all kind of software systems due to their dynamic and distributed nature. End-to-end testing techniques are one way to test the overall behaviour of web application. However, this kind of testing is often neglected since they are the most expensive level of testing.

The work in this thesis is motivated by a real-world problem in a Web Experience Management Platform (WEMP). Under this setup, websites are even more error-prone because multiple stakeholders can modify the website. In an e-commerce web site, where a minute downtime of web-front can interrupt the business, the impact of software failure is even more serious. A better testing strategy will result in a robust solution, improve availability and monitoring, reduce unpredictable behaviours. All of which can be converted into customer business value.

1.2 Research Question

The aim of the thesis is to apply End-to-end testing to Frosmo - an example of WEMP. A study is conducted on current approaches in end-to-end testing and the output is a suitable approach for the existing CI/CD pipeline. After implementation, the improvements and shortcoming of employing end-to-end testing in a WEMP will be reviewed. These are the three research questions that this thesis is seeking to answer:

RQ1: What are the requirements for the end-to-end testing pipeline in the case project?

RQ2: How to implement End-to-end testing in WEMP?

RQ3: Does end-to-end testing implementation meets the requirements in RQ1?

1.3 Structure of the Thesis

The structure of the thesis is as follows. Chapter 2 reviews existing literature about web application testing and end-to-end testing, as well as describe how end-to-end could fit into a WEMP. Chapter 3 elaborates research method and research question that the thesis seeking to answer. Chapter 4 describe the current state of the project and requirements gathering result. Chapter 5 provide detail on the implementation. Chapter 6 evaluate the implementation and Chapter 7 discuss the research questions. Chapter 8 concludes the findings and summarize the research.

Chapter 2

Background

2.1 Web Application Testing

Web applications are growing multi fold in popularity nowadays. When we carry out any task, most likely we opt to interact with a web application. They can assist us in work (calendar, email, task management), entertainment (social media, video, blog) or personal tasks (banking, shopping, doctor check-up). Web application are ubiquitous because they are available 24/7 for everyone with internet access and a web browser. Development and maintenance of web application are streamlined since the content of the website resides on server. Any changes to application can be updated in one place and automatically reaches all of its audience. In order to sustain their uptime and high reliability, web application require a competent testing strategy. Although there has been progress, the dynamic nature of web application still provokes a lot of challenges.

Web applications are defined as distributed systems with client/server architecture, where the clients are web browsers and the servers are the web application servers [1, p. 156].

Figure 2.1 show a simplified version of the web application architecture. In this architecture, the servers and data stores can be implemented as multiple instances, with different organizing scheme, developed in several programming language and deployed in multiple environment. The client browsers also varied in end-user type of browsers (and possibly multiple versions), devices (mobile and desktop view) and environment (operating system, location, language, etc.).

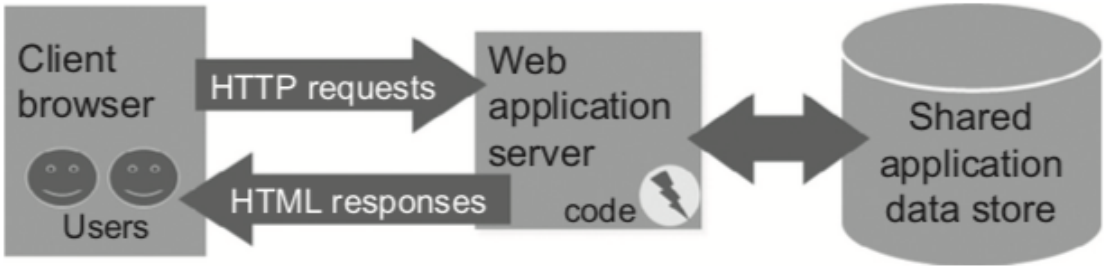


Figure 2.1: Web application architecture

Challenges

With such versatility, designing a testing suit, which cover all of possible combination of parameters is not an easy task [2]. Web application development iteration is usually much shorter than traditional software, which requires an efficient strategy, preferably high level of automation [3]. Furthermore, any error, downtime of website can directly affect a large proportion of users, cause heavy damage to business, especially in e-commerce field.

Sampath and Sprenkle [1, p. 155] identified the following as challenges in web application testing:

- A fast development cycle.
- Distributed architecture: Faults can occur in the server or client-side components or in the integration of these components.
- Multiple languages: Web applications are often written in multiple programming languages. HTML and CSS are a mark-up and style sheet language, respectively, that require different validation techniques.
- Multiple outputs: The results can be seen in the browser in the form of HTML document or changes in the application data store, in email messages, etc.
- Dynamic behaviour: Some types of web applications have code generated on the fly. Purely static analysis techniques may not fully be able to test the web application.
- Cross-browser, cross-platform compatibility.
- Large, complex code base: The distributed architecture of web applications necessitates a rather large code base that is often difficult for

programmers to understand. Web technology also evolves at fast pace, making the development more complex by using multiple coding and development frameworks.

2.2 Automation testing

Testing can roughly be divided into manual and automated testing. Manual test is where a human tester plays the role of an end user and executes the features of the System Under Test (SUT). Automated testing means the automation of those activities. To keep up with the pace at which application are developed and deployed today, automated testing has been playing a crucial part in web application testing. However, automating tests in web application is a very delicate process due to characteristics mentioned in 2.1. Approaches and tools on automation testing has come a long way during the last 10 years, but they are still having a lot of issue such as high upfront cost [4, p. 109], maintenance issues [5] and lack of skilled people [6].

Regarding the current state of automated web application testing, these are some noteworthy points under the scope of this thesis:

- There is still a gap between knowing the technical aspects of the process and applying the solution at an organization [3]
- Selenium is the most popular E2E test automation solution. Recently, there are other tools and framework for E2E testing, but studies and practice on them are lacking [1, 7].
- There is a tendency of moving web application to the cloud, hence, implementing test in the cloud should be more focused [2]

2.2.1 Test Automation Pyramid

Automation testing enable inexpensive re-execution of test, which allow better coverage and increase software quality confidence. In low-level testing, such as unit level, test cases focus on specific behaviour in code logic. Therefore, they are actually more straight-forward to test programmatically. However, automation still cannot entirely replace manual testing and planning for a balance between manual and automation testing is not an easy task. The testing pyramid - a widely adopted model by Mike Cohen [8, p. 312] proposed the amount of automated testing we should be doing at each testing level. In his original idea, the test pyramid consists of three layers that a test suite should be consist of (from top to bottom):

1. UI Tests
2. Service Tests
3. User Interface Tests

In this model, the higher the layer, the more integrated and more expensive the tests are. Thus, most tests can and should be automated, and we should implement more tests at lower level.

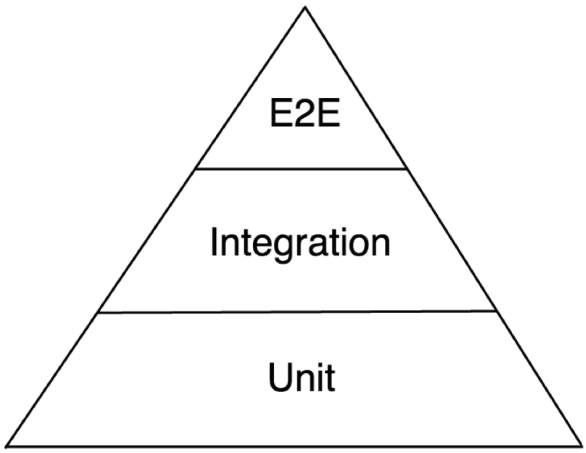


Figure 2.2: The test automation pyramid

2.3 End-to-end testing

The term "End-to-end" means "from-the-start-to-finish". E2E testing is defined as a type of testing that validates the whole application stack and architecture, including integration of all services and component that may or may not be belong to the SUT [9, 10]. The following sections consider where this testing approach can be used:

In Testing Levels

Tests in software development process could be categorized into testing levels. Testing levels define the level of abstraction of the tests. In the days of sequential software life cycle models, they are different stages in the software development life cycle where testing is conducted. By separating them into

Level	Summary
Unit Test	Test individual component
Integration Test	Test Integrated Component
System Test	Test the entire system where all components are integrated
Acceptance Test	Final users test in consumer environment. These tests can be seen as functional testing performed at system level by final users or customer [7]

Table 2.1: Levels of testing

different layer, we avoid overlap and repetition of tests. In a popular software life cycle model: V-model [11], there are four level of testing (Figure 2.1)

As the name implies, E2E approach should be at a high level of testing, where the whole system is tested. In fact, E2E testing is considered a special type of system testing where final user is simulated [12].

In Agile testing Quadrant

Agile is a software development approach, which promotes self-organizing and cross-functional teams. It advocates rapid response to change in development [13]. In Agile development, project is broken into small iterations. Each iteration goes through all function of a traditional development project: planning, design, development, testing. The output of each iteration is a working product, which can be demonstrated to stakeholders. This reduces wasting resource in planning and design, minimized overall risk and allow the product to adapt to changes better.

Since testing in Agile project is no longer a separate phase by itself, testing levels overlap with each other. Gregory and Crispin [14, p. 102] found it helpful to use the Agile testing quadrants model (Figure 2.3)

On the x-axis, we divide the matrix into test that support the team and tests that critique the product. On the other axis, we divide them into business-facing and technology-facing tests. As a result, the tests are divided into four quadrants:

- Quadrant 1: Unit and Component tests, support the team at technical level, they let programmers confidently write code.
- Quadrant 2: Support development team at a higher level, define external quality that customers want, hence the business-facing aspect.

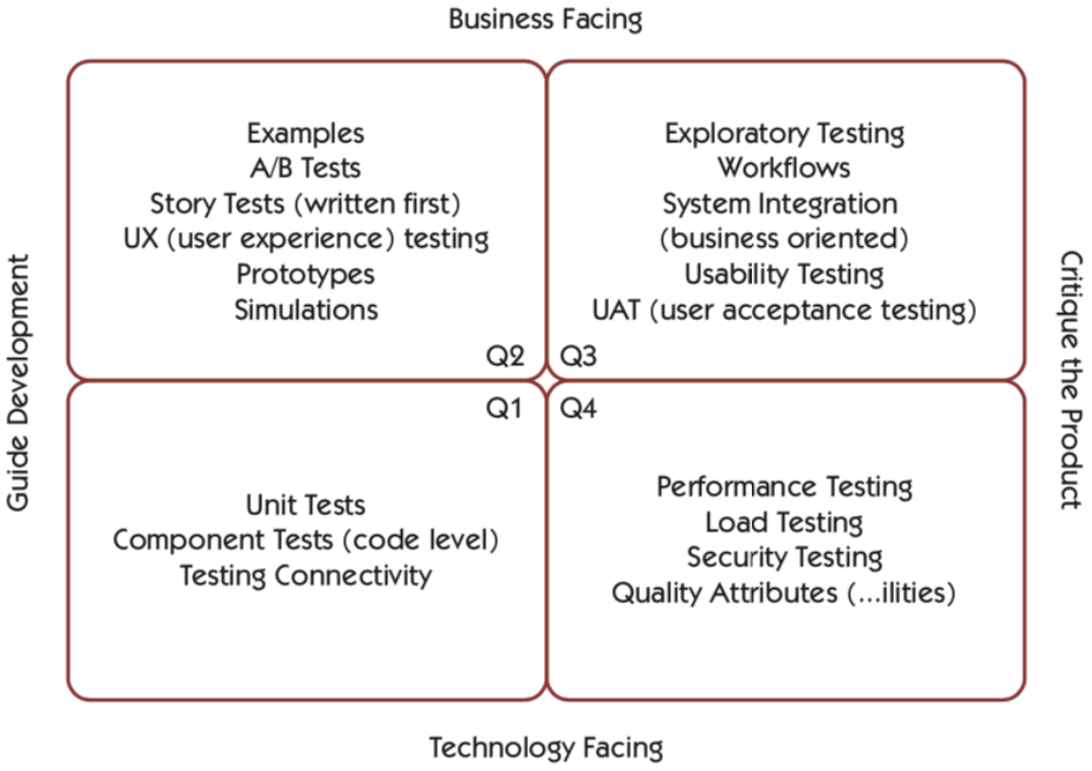


Figure 2.3: The Agile Testing Quadrants

- Quadrant 3: These tests critique the product, cover paths outside the specifications and often emulate or involve real users.
- Quadrant 4: Also on the side of critique, but technology-facing. These tests include performance, security and "-ility" testing.

E2E approach usually used in User Acceptance Testing, which belongs to the 3rd quadrant [14, p. 189]. They demonstrate the desired behaviour throughout every part of the system.

In Test Automation Pyramid

In the original version, the three level in Mike Cohen's pyramid are: Unit, Service and UI (section 2.2.1). But in the subsequent comments, he agreed with others that the UI level should be called "End-to-end" tests [15]. By "UI" he rather refers to the testing tools, but "End-to-end" is more accurate

term for what actually the tests are. E2E testing located in the highest level of the testing pyramid. Which means it is the most top down and comprehensive test on the application (from the UI down). The term "End-to-End" is also broadly accepted in practice, e.g., Gitlab [16], Ham Vocke [17]).

UI testing and E2E testing are not the same, but rather orthogonal concepts [17]. In a web application, E2E testing is almost always UI testing, seldom they can be REST API test, and not all UI testing is E2E. Testing the UI can be some unit test in front-end JavaScript code, without requesting any resource from backend. While E2E testing that involve UI also involve all backend functionalities. Modern frontend frameworks like React, Angular, Vuejs provide their own tools of writing unit test against UI element.

2.3.1 Types of E2E testing

There are two types of E2E testing: vertical and horizontal:

- The type most commonly used is horizontal: which goes through user workflow or transaction across multiple applications from start to finish to ensure that each process occurs as expected [18]. In the testing pyramid, horizontal E2E happen at the top-most level, testing the whole system the way it will be used, as a real-world user [12].
- In vertical E2E: Test will be implemented on all levels of the pyramid, from unit tests then continuing up to tests to the Integration and UI layer. [18].

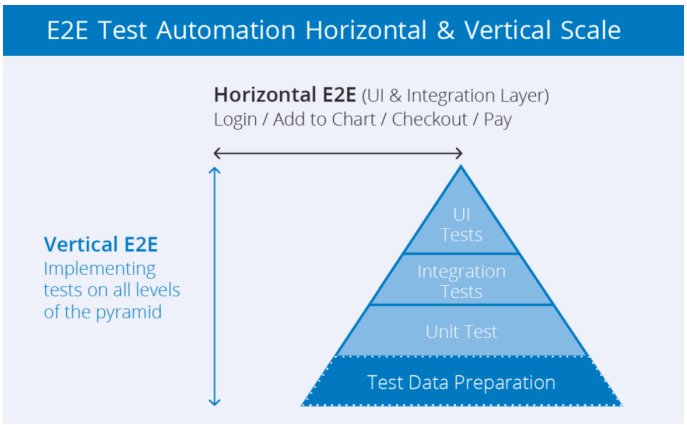


Figure 2.4: Vertical and Horizontal E2E testing [18]

2.3.2 Challenges

E2E testing may seem to be a good way to gain confidence for the whole SUT in a simple case. Nevertheless, since these tests drive through UI and touch every component in the system, they are expensive and time consuming to run [19]. Automation of test execution is one answer to this [20]. Moreover, E2E tests tend to break often. Literature suggest that simple modification to the SUT result in 30 - 70% changes to the test cases [4], which increase maintenance cost. E2E test are expensive to develop yet also lacking robustness. Recent study on E2E testing, therefore focus on automatically creating test or repair test cases [21, 22, 23, 24, 25] Another solution could be to limit test cases to the most crucial customer journey [17]. In an e-commerce site, for example, this can be reduced to search and product purchase.

2.3.3 Automating E2E test in Web application

On top of E2E challenge discussed in section 2.3.2. testing distributed system and web application in particular are difficult with their heterogeneous and distributed nature (discussed in section 2.1). According to a survey conducted by Lima and Faria [26], there is a gap between the current and the desired status of test automation for distributed heterogeneous system, and we should prioritize automation for this type of systems. In the scope of web application, the main mechanism used for E2E testing is black box testing based on the concept of test scenario [20], which is a sequence of steps performed on the UI of web application, then retrieve information to compare with the test oracles [27]. Automation usually means automated execution of those interactions and assertion of information.

2.3.4 Classification

There are various tools and practices regarding automation E2E testing of web application. Garcia et al. [7] list these categories of tools and frameworks:

- Facilitate test case creation with record and replay: Selenium IDE [1]
- Writing code or script: Selenium Web Driver [2], Appium [3], CasperJS [4], etc.

[1] https://selenium.dev/selenium-ide/
[2] https //selenium.dev/documentation/en/webdriver
[3] http //appium.io/
[4] https //github.com/casperjs/casperjs

- Facilitate cross-browser execution of tests across platforms: BrowserStack [5], SauceLabs [6].

Leotta et al.[28] classify testing approaches base on two criteria: test case construction and element localization.

Test case construction

Regarding test case construction method, there are two main approaches:

1. Capture-Replay (C & R) Web Testing: record the actions performed by the tester on the Web User Interface (UI) and generate a script that provides such actions for automated re-execution.

2. Programmable Web Testing: Tests are written with the help of specific testing frameworks, which supports programming of the interaction with a Web page and its elements.

Element localization

In order to interact with web UI elements in UI test, it is necessary to locate different elements such as input fields, titles or buttons. One of the main difference between methods of automated UI testing is how these elements are located:

1. Coordinate-based localisation: during capturing phase, the tools just record the screen coordinates of the Web page elements and then use this information for tests re-run. "This approach is nowadays considered obsolete, because it produces test cases that are extremely fragile" [28].

2. DOM-based localisation: Web page elements are located based on Document Object Model. For example, the tools Selenium IDE and WebDriver employ this approach.

3. Visual localisation: An emerging approaches, which makes use of image recognition techniques to identify and control GUI components. E.g., SikuliX [29].

[5]https //www.browserstack.com/
[6]https //saucelabs.com/

Analysis of the approaches

Leotta et al. [20] concluded:

1. Regarding test case construction methods: Programmable tests approaches involve higher development effort (32% and 112%), but lower maintenance effort (16% and 51%) than C%R approaches. However, programmable method enables defining parametric and repeated test scenarios which add up to their advantages.

2. Regarding localizing web element: DOM-based approaches are better in overall. They are more robust, require lower development and evolution cost, lower execution time than visual locators. Visual locators are only more suitable in some specific cases, where visual appearance of application remained stable across release.

2.4 E2E testing in Continuous Development

2.4.1 Continuous Development

Lean concept of software development and recent emphasis on DevOps realize that the integration between software development and its operational deployment needs to be a continuous one. Hence, the emergence of many activities all tied to the term "continuous": continuous planning, continuous delivery, continuous verification, continuous monitoring, etc. The most popular among them are Continuous Integration (CI) and Continuous Deployment & Delivery (CD).

CI can be defined as a practice of continuously merging code changes and validate these changes through steps of compiling, validating, compliance checking and deployment preparing. It is typical that the process happens frequently and automatically, allowing integration problems to be discovered and solved as soon as possible [30].

CD is the the next step after CI, referring to the practice of continuously deploying software build automatically to some environment, but not necessarily actual user. A closely related term is Continuous Delivery, which implies Continuous Deployment and extend its meaning that software is continuously ready for release and deployed to actual customers [31].

Together, CI and CD practices can be called CI/CD pipeline or deployment pipeline [32, Chapter 5].

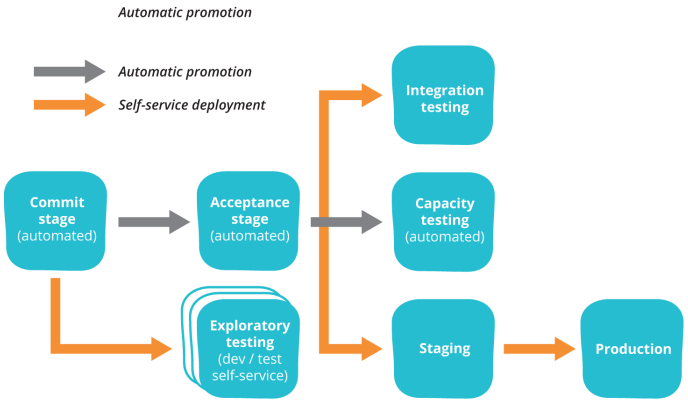


Figure 2.5: Deployment Pipeline

2.4.2 Automated E2E testing practice

Automated E2E testing is one popular practice in CI/CD [32, 33, 34]. E2E tests help to imitate the usage of the system by the end user with all dependencies and integrations and the execution of these tests should be automated to conform with Continuous Development idea. Humble and Farley [32] argued that automatic acceptance testing is "essential". For short project with small team of four or fewer developers, they advised to start with a few E2E tests as part of a single-stage CI process. Applying the guideline above, Kim et al. [34, Chapter 10] also deem it not necessary to automate all the tests since E2E tests tend to break easily, which gradually result in developers ignoring test failure or even abandoning test writing. They would rather be having a few but reliable tests.

2.4.3 Frameworks and tools

This section provides some of the popular tools used for web E2E testing. There are three main criteria for choosing this list:

- Runnable on Linux server
- Cross-browser support
- Can be integrated with GitLab CI.

The most popular Web UI testing tools and their comparison are shown in Table 2.2. The list is summarized by searching for relevant articles mainly on Google and Google Scholar with keyword "Web End-to-end testing tools"

and "Web UI testing tools". The terms were chosen based on similarity between tools for E2E and UI in web testing. The list presented should not be considered a comprehensive sample of all framework and tools related to automated E2E testing. However, the findings provide a sensible overview of the different solution available and their benefits and disadvantages. For example, almost every testing tool in the market facilitate scriptable testing (programmable test). Capture and Replay method is also very popular. Although most of them are not presented in the table since the project in case focus more on programmable approach. Capture and replay feature usually shipped in complete-software-solution with less flexibility. As identified from literature, Selenium proved to be the most popular set of tools. Selenium IDE for test recording and Selenium WebDriver for browser control are employed by many testing solutions. Visual localizing method are available in some of the tools. But most of them are trademarked technology developed independently (eggPlant Functional [35], Maveryx [36]), not adopting open-source project like Sikuli [29].

This chapter show a context of E2E testing in web application as in existing literature and reviewed approaches and tools in applying it in CI/CD. These insights are necessary in order to achieve the objective of the work in the next chapters.

Name	**Browser**	**Visual localizing**	**C & R**	**License**
Cypress	Chrome	No	No	MIT (Test Runner) and Proprietary (Dashboard)
eggPlant Functional	IE, Firefox, Safari, Opera, Chrome	Yes	Yes	Proprietary
Gauge	cross-browser	No	No	GNU GPL v3.0
Katalon Studio	IE, Firefox, Chrome, Safari, Opera and any modern browser	No	Yes	Proprietary free
Maveryx	IE, Firefox, Chrome, Safari, Opera and any modern browser	Yes	No	Proprietary
Nightwatch	IE, Edge, Firefox, Safari, Opera, Chrome	No	No	MIT
Redwood HQ	Chrome, Firefox, IE	No	No	GNU GPL v3.0
Robot Framework	IE, Firefox, Chrome, Safari, Opera and any modern browser)	No	No	Apache License 2.0
Sahi	IE, Firefox, Chrome, Safari, Opera and any modern browser)	No	Yes	2 versions: open-source and Proprietary
Selenium	cross-browser	No	Yes	Apache License 2.0
SOAtest	cross-browser	No	Yes	Proprietary commercial software
Watir	IE, Firefox, Chrome, Safari, Opera, Edge	No	No	MIT license

Table 2.2: Comparison of web UI testing tools

Chapter 3

Methods

Design science research method [37] is used in this thesis. Since the research is about constructing an artefact - the E2E test for the project, it would fit well into the model. The research process is divided into iterative steps: problem identification and motivation, objectives and solution definition, demonstration, evaluation and communication. Peffers et al. described those activities with examples in more details in the guideline [38]. The process is elaborated in Table 3.1

Identify Problem

The study was conducted on Frosmo, which is a WEMP. The platform is described in more details in 4.1. With Frosmo, customer website can be developed, modified indirectly using a JavaScript tag. Frosmo platform also provide monitoring and flexible control over those modifications. While providing convenience, this system introduces brittleness to the website since multiple side can modify the website. Most of Frosmo works relied on the front-end of the website, so any change in the UI could result in unexpected behaviour. There are automated unit testing, integration testing and manual acceptance tests on each feature released from Frosmo. However, the test only covers Frosmo pipeline, any development from Customer side is not considered. Due to the problems mentioned above, it was seen as important to implement E2E testing so that the User-facing side is verified. Automation should be employed since development from both sides are released regularly and asynchronously. The solution could be first developed for a customer project, but then should be reusable for other case project inside the company.

Step	Content
Identify problem & motivation	How does E2E testing could benefit Frosmo case?
Define objectives of a solution	Which E2E framework are suitable for the project and should be implemented?
Design & development	Design and implement an artefact based on previous two steps
Demonstration	Introduce prototype to team and integrate it to development process
Evaluation	Observe its effectiveness. Feedback from team and customer.
Communication	Provide research result report and presentation

Table 3.1: Design Science Research Methodology

Define objectives of a solution

The solution is to implement a E2E testing setup and actual tests for a customer case project in Frosmo. To define the objectives, we used a requirement elicitation process to gather information: conducting interview with product managers, developers and QA engineer; ask them for their expectation of the solution. The result of the process will be described in detail in Section 4.3.

Design and development

Chapter 5 will elaborate the design and development step. From the requirements, we will propose the solution design and integrate it into the current CI/CD pipeline. A customer case project will be chosen to implement with E2E test and introduce testing into the current workflow.

Evaluation

The implementation result will be discussed: what features have been implemented, quality of implemented system and feedback from stakeholders. The evaluation step will be described in Chapter 6

Communication

Research result are documented with internal documentation tool and presentation are conducted to introduce findings among the company.

Chapter 4

Case project analysis

4.1 Web experience management platform

Frosmo solution - WEMP is a new solution for developing, improving website functionality. The idea is generally simple, a JavaScript tag placed directly in the customer web page HTML code will handle all the connection from the platform to the website. Then, the consultancy company will provide customers with solutions, statistics, and development, visualized by modification deliver directly to the website front-end. The customer can also monitor and adjust these modifications using the platform panel.

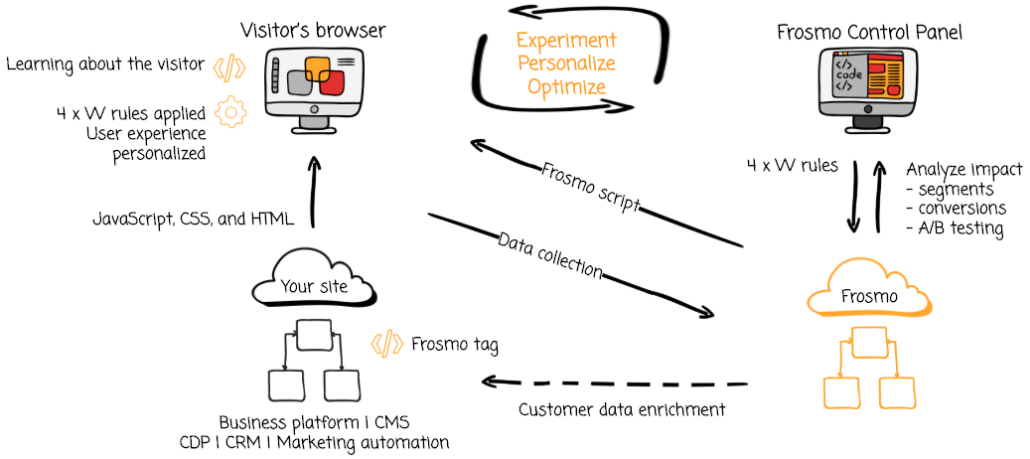


Figure 4.1: Frosmo Overview

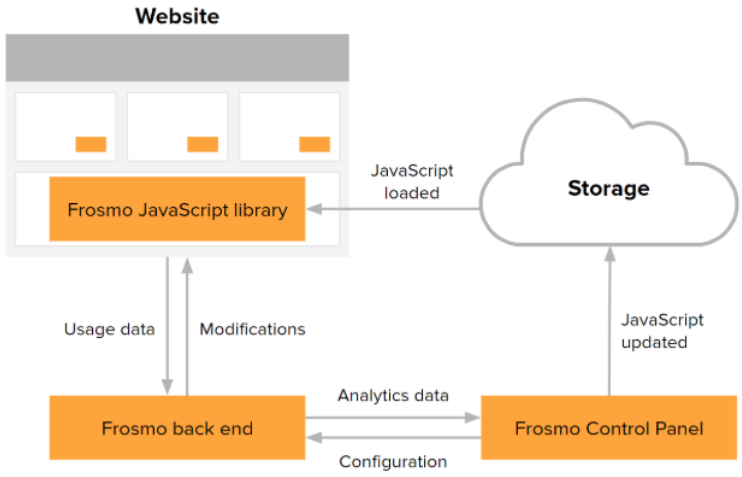


Figure 4.2: Frosmo Platform architecture and information flows

Figure 4.2 is an overview of Frosmo Platform architecture. It consist of three main components:

- **Frosmo JavaScript library** handles the modifications to the site, manages segmentation, collects usage data, and fetches content to display from the back end.
- **Frosmo back end** stores the usage data collected by the Frosmo JavaScript library and processes the data for reporting and analysis purposes. The back end also stores operational data related to modifications, segments, and other configurable resources.
- **Frosmo Control Panel** (FCP) manages the Frosmo JavaScript library (and thereby how the site is modified and improved) and pull analytic data from the Frosmo back end. Modifications are written on FCP by developers or client developers.

In addition to modifications, more sophisticated solutions will be developed in a separated customer repositories hosted on GitLab. Output script will be placed in customer website in a second script tag called textbfCustomer Script along side with Frosmo JavaScript Library.

4.2 Development Process

A typical feature development process is as follows:

- PM and Customer discuss new feature and create development tasks.
- Developers work on the feature, test and do code review within the development team.
- The reviewed feature is passed on to QA team, who will carry out manual UI tests to verify the feature on different browsers and mobile devices.
- The feature is then delivered to the customer website. Customer and PM are usually in charge of making small updates, managing and monitoring.

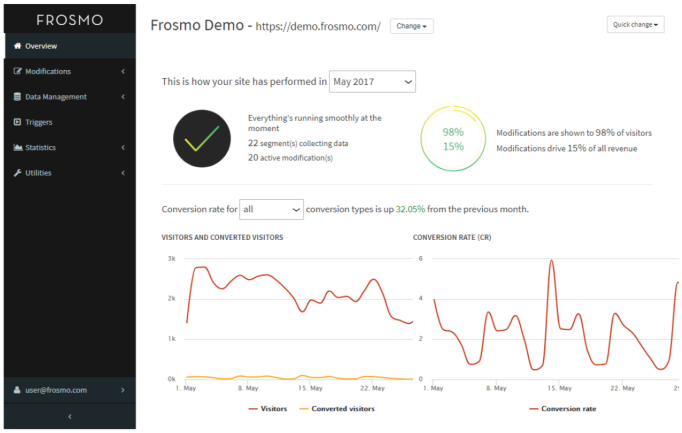


Figure 4.3: Frosmo Control Panel

The following subsections will examine technical aspects of development work flow.

4.2.1 Code base

In a typical platform client case, there are two separated code base:

- FCP Modification: Hosted in FCP. Can be edited by Clients and Frosmo developers and PMs. No version control and CI/CD.
- Customer Script: Hosted in GitLab. Can be edited by Frosmo developers. Has CI/CD.

4.2.2 Workspaces

Workspace is an FCP feature, allows separation of development in FCP. Developer can create both new features and edit existing ones in a workspace without affecting the live content of the website. Reviewer can also safely preview the workspace content on the site. Customer Script can also be deployed to workspace for integration test. Workspace is used in development work flow similar to develop branch.

4.2.3 CI/CD pipeline

Customer Script in GitLab follow a CI/CD pipeline. Any changes pushed to a ticket branch will undergo automated unit test then built to workspace for review. After feature is manually verified by PM, QA and code-reviewed, it can be deployed.

Development

- After prerequisites work has been done, a ticket is created by PM. This ticket has a unique *ticket-id*.
- Developer create a Workspace with the *ticket-id* in FCP
- if Customer Script is needed, developer will also create a branch *ticket-id* in GitLab repository.
- Developer start to develop.

Testing/Staging

- Developer pushes to branch via git.
- Unit tests are run by GitLab against custom code.
- PM, QA and Customer review the feature through the workspace.
- Code reviewed by developer.

Deployment

- Customer Script is deployed.
- Workspace is deployed.

Current pipeline employs automated Unit Test using Jasmine and manual Integration Test. However, it does not account for modifications. There is

no version control nor CI/CD for FCP modifications. Other stakeholder such as clients and PMs can edit modification via FCP, which is outside of verification process. In addition, client developer can introduce feature to website that may break old modifications. To cope with these uncertainties, E2E automated testing should be implemented.

4.3 Elicitation of Requirements

In order to gather requirements for the project, a requirement elicitation process [39] was carried out. The stakeholders are segmented in order to avoid the pitfall of treating them as one homogeneous group. Then, interviews are conducted with candidates from those segments to discover and extract the requirements. Next, development team came up with implementation ideas to meet those requirements. Finally, the list of ideas is presented in a workshop in order to prioritize requirements and to choose the ideas to be implemented.

4.3.1 Segmentation

There are three stakeholder segments in this project:

- Developers
- Product Mangers
- QA team

The first segment is the developers in Frosmo Customer Success department, who use Frosmo Platform and implement clients work. Product Manager is the person who take responsibility of each client, manage client tasks and facilitate work flow between client and developer team. QA team is a separated team in Frosmo, who carried out QA work for all clients.

4.3.2 Elicitation

The author of the thesis interviewed two developers, one manager and one QA engineer about their expectations for E2E test implementation. It was done to discover and elicit requirements for E2E test and how it can be integrated into current development pipeline. The result are requirements, ideas and practices that can be divided into three categories: Source Control, Build and Test Running, Monitoring. After the interviews, the collected information is then analysed and a brainstorming session is organized within

E2E testing project team to refined the data and discussed ideas to resolve the requirements. The outcome of these sessions is a sets of implementation ideas for each requirement (section 4.4.1 - 4.4.3).

Finally, a workshop was held with people from all segments to prioritize requirements. Result is a prioritized requirements table (section 4.4.4).

4.4 Result

4.4.1 Test Programming

Use Version Control System to store E2E test code

Description: Current practices at the case company use GitLab as Git-repositories platform for all project code. GitLab proved to be a robust repositories platform. Preliminary research show that GitLab CI/CD functionality is sufficient for the case project. Each client project has their own code repositories. Unit test code is stored inside client repositories.

Implementation Ideas: Use GitLab as VCS for E2E test code since teams already have experience with Git and GitLab.

Separated repositories for test code

Description: There are two ideas on repository: Separated repository for E2E tests or Same client project repository. Same repository is the first reasonable choice since it avoid code partition. Meanwhile, separated repository is easier for E2E development team to pilot the project without affecting clients development.

Implementation Ideas: First, pilot one or two client project with separated repositories. After validation and refinement, tests can be moved into client project repository.

Support Peer review

Description: Branching model is used in the company to support isolating development work and peer review. However, each team decides on itself how to branch on features and tasks.

Implementation Ideas: Separated test repositories for each client. Branches should be used for isolation of work. Peer review should be done before merging branch to master using GitLab merge request.

4.4.2 Test Running

Run test on different types of browser

Description: Test should be ran on multiple browser type (Chrome, Firefox, Safari, etc.).

Implementation Ideas: Prioritize choosing E2E testing framework that support multiple browsers.

Test run automatically

Description: Test run should be automated to provide quick feedback and effectively guard against uncertainties in WEMP.

Implementation Ideas: There are different opinion on whether tests should run on commit or to be scheduled at fixed time. The decision could vary, based on test execution cost and client needs. Therefore, both approach should be made available via CI system.

Test environment based on VMs

Description: Test environment should be easy to set-up and highly available.

Implementation Ideas: use Docker to run test isolated in Amazon Web Service (AWS). Case company is migrating to Docker and AWS cloud service for most of the schedule tasks. AWS makes it straightforward to monitor cost and other indicators. Docker is easy to set up, tear down and provide customized test environment if needed.

4.4.3 Monitoring

Code coverage is measured

Implementation Ideas: Choose test framework that support measuring of code coverage.

Support active alert

Description: Any issues found should be raised as soon as possible to increase feedback loop. Case company is currently using Kibana [40] for error tracking. A physical PC/monitor is set up in each team to display ongoing status of each clients.

Implementation Ideas: Kibana integration, alert email or Slack channel notification.

Test data for to non-technical stakeholders

Description: Test reports should be intuitive and accessible for non-technical users.

Implementation Ideas: Choosing test framework that support HTML reports or graphical statistic. Store test reports in FCP or Amazon S3, making it highly accessible for Clients, PMs and Marketing team.

4.4.4 Prioritized Requirements

The results of the elicitation process are summarized in the Table 4.1. The column "Importance" answers the question "How important is implementation of the requirement for the case project". The result value is the average of grades given by participant of the final workshop where 0 means "not importance at all" and 10 means "most important".

The results of the workshop help the team to focus on features that are most concerned by stakeholders.

Requirements	**Category**	**Importance**
Use GitLab as VCS	Programming	10
Setup test environment with AWS	Running	9
Test data storage in Amazon S3	Monitoring	8
Run test on schedule	Running	8
Run test on different type of browsers	Running	7
HTML reports	Monitoring	7
Separated repositories for pilot projects	Programming	6
Alert via Slack	Monitoring	6
Use Docker as test container	Running	5
Branching and merge request for peer review	Programming	5
Same repository as client project for E2E test	Programming	3
Run test automatically on commit	Running	3
Alert via Kibana integration	Monitoring	2
Test data storage in FCP	Monitoring	1
Code coverage is measured	Monitoring	1
Graphical statistic	Monitoring	1
Alert via email	Monitoring	0

Table 4.1: Priorities of Requirements

Chapter 5

Implementation

5.1 Robot Framework

For writing and running E2E test, Robot Framework was selected. The selection process is out of the scope of this thesis. These following points contribute to the decision of choosing Robot Framework:

- Supports running test on all major web browser (Chrome, Firefox, Safari, etc.).
- Open source software.
- Consists of various internal and external test libraris and tools for creating tests.
- Creates uniform and reusable test scripts.
- Easy to read reports and logs in HTML format.

5.1.1 Architecture

Robot Framework has a modular architecture that is easy to extend with libraries and tools. Test data and files can be defined and imported across directories to create a nested structure of test suites depend on the scale of the project. Figure 5.1 depicts Robot Framework modular architecture.

5.1.2 Setup

Editor

RIDE [41] is a lightweight and intuitive editor for Robot Framework test data. RIDE make it easy for QA engineer or non-technical personel to edit

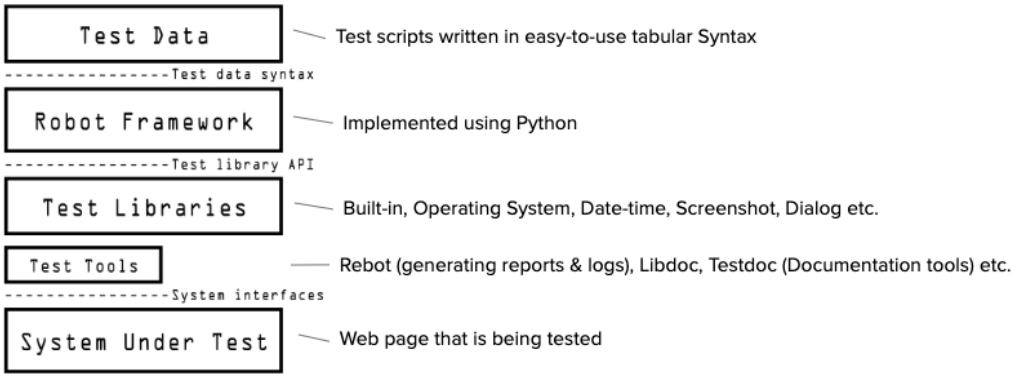


Figure 5.1: Robot Framework Modular Architecture

test code. Since Robot Framework is written in tabular Syntax, normal text editor can also be used. For example, most developers use Visual Studio Code [42] to edit test code.

Library

Since Robot Framework is only a generic automation framework, a Web testing library is needed for it to function as a web testing tool. Therefore, the following libraries are added.

- **SeleniumLibrary**: web testing library uses the popular Selenium tool to control web browser.
- **Webdriver**: After installing the library, it is required to install specific browser drivers for all the browsers that need to be tested. Selenium Webdrivers cover all major web browsers and is available for download at SeleniumHQ Downloads Page

5.1.3 Test case

There are several ways to write test cases. Robot Framework describes two styles of writing: Keywords-driven and Data-driven. Test cases that describe some type of workflow could be written in keywords-driven style. The data-driven style is when we want to test a scenario with different input data. The project case will employ keywords-driven style with the idea of verifying only the most common path.

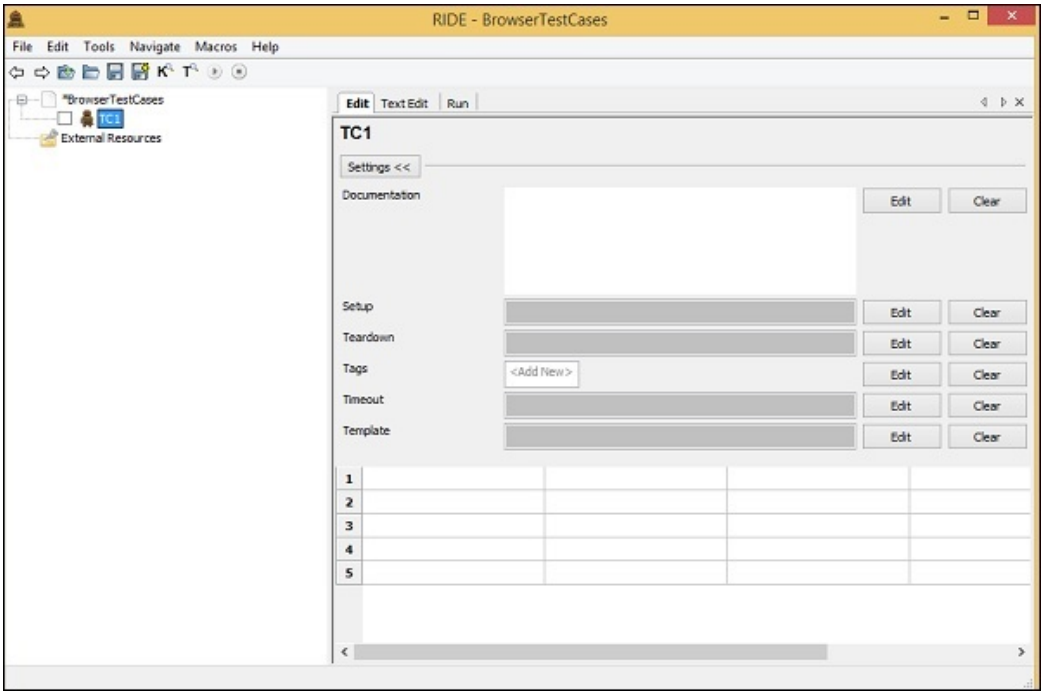


Figure 5.2: RIDE - Test data editor for Robot Framework

Writing Tests

Figure 5.3 shows a simple test case written in keywords-driven style. Variables are used to hold a value, which can be used in test cases or user-defined keywords. In the example Figure 5.3, "test_variables.robot" defines two variables to use in "test_keywords.robot".

There are two type of keywords, user-defined and library-defined. In the example, SeleniumLibrary is included, which enables usage of Selenium WebDriver keywords, e.g. Open Browser, Close Browser. Other user-defined keywords can be created like in "test_keywords.robot".

Running Tests

Normally, Robot Framework test can be run from terminal:

```
robot sample-test.robot
```

In this project setting, test code is run in docker container. An modified version of Robot Framework image *ppodgorsek* from Docker Hub [43] will be

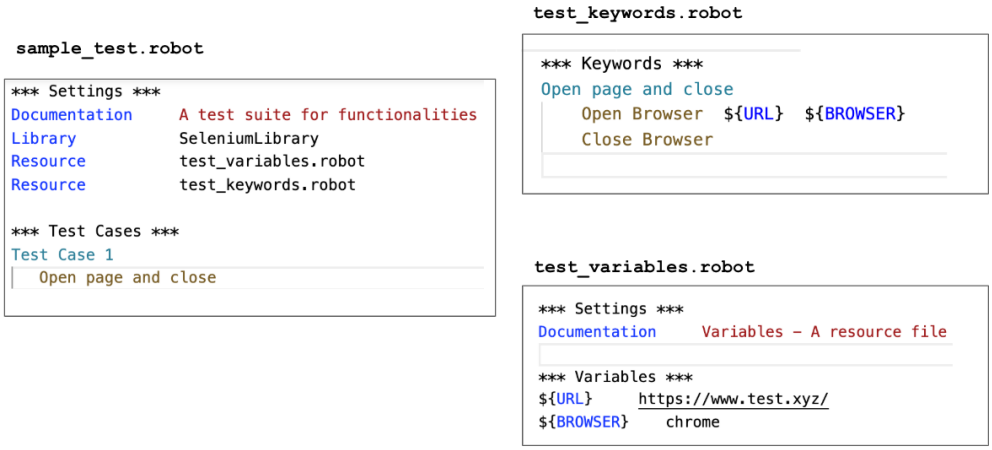


Figure 5.3: An example test case written in keywords-driven style

used. GitLab runner will start a docker-in-docker (DinD) and run a docker container to execute robotframework test. Test piepline is configured to run the test daily at 12pm.

```
docker run \
   --shm-size=1g \
   -e ROBOT_OPTIONS="-v BROWSER:chrome --suite Tests.DNA" \
   -v `pwd`/reports/DNA:/opt/robotframework/reports:Z \
   -v `pwd`/tests:/opt/robotframework/tests:Z \
   ppodgorsek/robot-framework:latest
```

Reports

A test run from Robot Framework will output a HTML test report similar to Figure 5.4.

The Robot Framework plays the main part in the case project E2E testing automation. Robot Framework runs the test, outputs the logs and reports. Test run will be executed by GitLab CI/CD in a Docker container. Test reports will be stored as artifacts in GitLab server and in Amazon S3.

5.2 Architecture

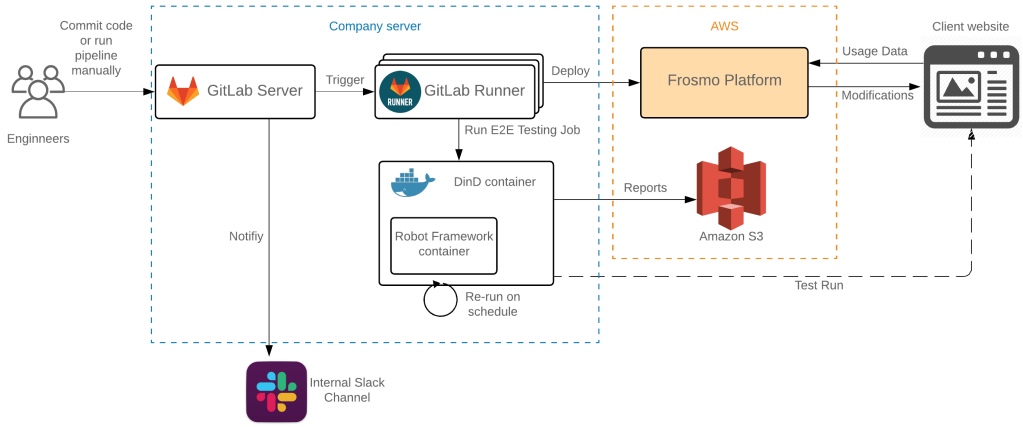


Figure 5.5: Pipeline components

The DevOps pipeline was designed based on preliminary analysis of the tools and workshop sessions. Existing technologies such as GitLab, Docker, AWS, Slack were taken into account and applied in the testing pipeline.

Pipeline components are shown in Figure 5.5. When engineers commit code to the GitLab server it will triggers GitLab Runner[1]. Depending on each project setting, the jobs will be run, usually consist of unit testing, build to workspace or deployment to production.

The new E2E pipeline will add a E2E testing job to the pipeline. Due to the cost of running E2E test, this job is scheduled to run everyday at midnight, rather than on each commit. The job will be run using a Docker container with required environment for running Robot Framework test. Docker-in-docker (DIND) was used to be able to run docker command in GitLab CI/CD [2]. DIND is a Docker image with all Docker tools installed. This is one recommended approach by GitLab to run Docker container with GitLab CI/CD. Figure 5.5 show that Robot Framework Docker container is running on company's premises. However, depending on project requirement, this setup can be moved to AWS if needed.

[1]https //docs.gitlab.com/runner/
[2]https //docs.gitlab.com/ee/ci/docker/using_docker_build.html

After execution, E2E testing report and logs will be saved as artifacts[3] in GitLab (with an expiration period of 4 weeks). Reports and logs are also copied to Amazon S3 bucket to be archived. Failed jobs on E2E testing job will be notified to a separated Slack channel.

5.3 CI/CD pipeline

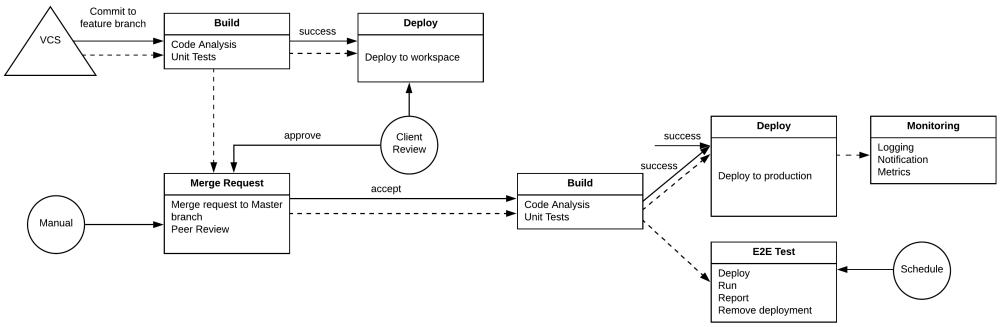


Figure 5.6: CI and CD pipeline

The CI/CD pipeline of the case project is described in Figure 6.1 using Ståhl & Bosch notation [44]. Conforming with Git workflow model, developer checks out the code to a feature branch for each feature development. The input of the pipeline is a commit to a feature branch. It triggers build job in GitLab, which runs code analysis using ESLint[4] and Unit Tests using Jasmine[5]. If the build job successes without any error, it triggers deploy to workspace. After deployment is done, feature can be submitted for Project Manager and Client review. After being approved by client, feature can proceed to merging. A merge request to master branch is submitted by developer, which then goes through a code reviewing process. Any issues raised from the reviewing should be fixed and changes should be committed to the feature branch, triggering build job, merge request update. This process will be repeated until code is approved for merging. The merge will triggers the build in master branch, after successful build, the code is deployed to

[3]https //docs.gitlab.com/ee/user/project/pipelines/job_artifacts.html
[4]https://eslint.org/
[5]https://jasmine.github.io/

production. E2E test are scheduled to run nightly to minimize the execution cost.

5.4 Creating test

The pilot client for the case project is **DNA.fi**. The product is their customer facing web store.

5.4.1 Project background

The client project **DNA.fi** applies Agile methodologies. Development tasks are divided into sprints. Each sprint is an iteration and has its own development phase and testing phase. Similar to a typical Agile project, requirements are given in the form of Epics and User stories.

User stories

In Agile development, an user story is an informal, natural language description of one or more features of a software system. User stories are often written from the perspective of an end user or user of a system.

Epic

Multiple user stories, which represent a big chunk of work usually compose an epic. Stories under an epic should have a common objective, e.g. a feature, customer request or other requirements.

5.4.2 Test cases

An analysis was conducted for the pilot client DNA.fi in order to identify keys features in the customer facing web store. There are six features/areas classified as meaningful for epic level:

- **Login**
- **Filter** Filtering product in the main shopping page
- **USP** Recommendation feature developed by Frosmo
- **Phone Purchase**
- **Package Purchase**
- **Internet Purchase**

Each epic consists of multiple test cases that cover different input and outcome of user journey on the web store. There are total of twenty test cases written for this case project. They are presented in Figure 5.7

5.5 Monitoring

The reports and logs from each run will be saved as artifacts in GitLab, which can be downloaded (with an expiration duration of 4 weeks). Reports are also copied to Amazon S3 bucket for storage. Notification on failed jobs will be sent to a dedicated internal Slack channel.

5.6 Implementation Problems

This section describe the problems that emerged during the development.

Personalized data within a test case

The first problem encountered was testing Frosmo recommendation on client website. In RobotFramework, each test case starts in a new incognito tab. This is good for data isolation, but makes it complicated for personalization. Some Frosmo recommendations rely on local storage to save data. While each test case will run on a incognito browser tab. Therefore, no personalized information is maintained. Any features required a streamlined flow of personalized data need to be run in one test case.

Fragile end-to-end test

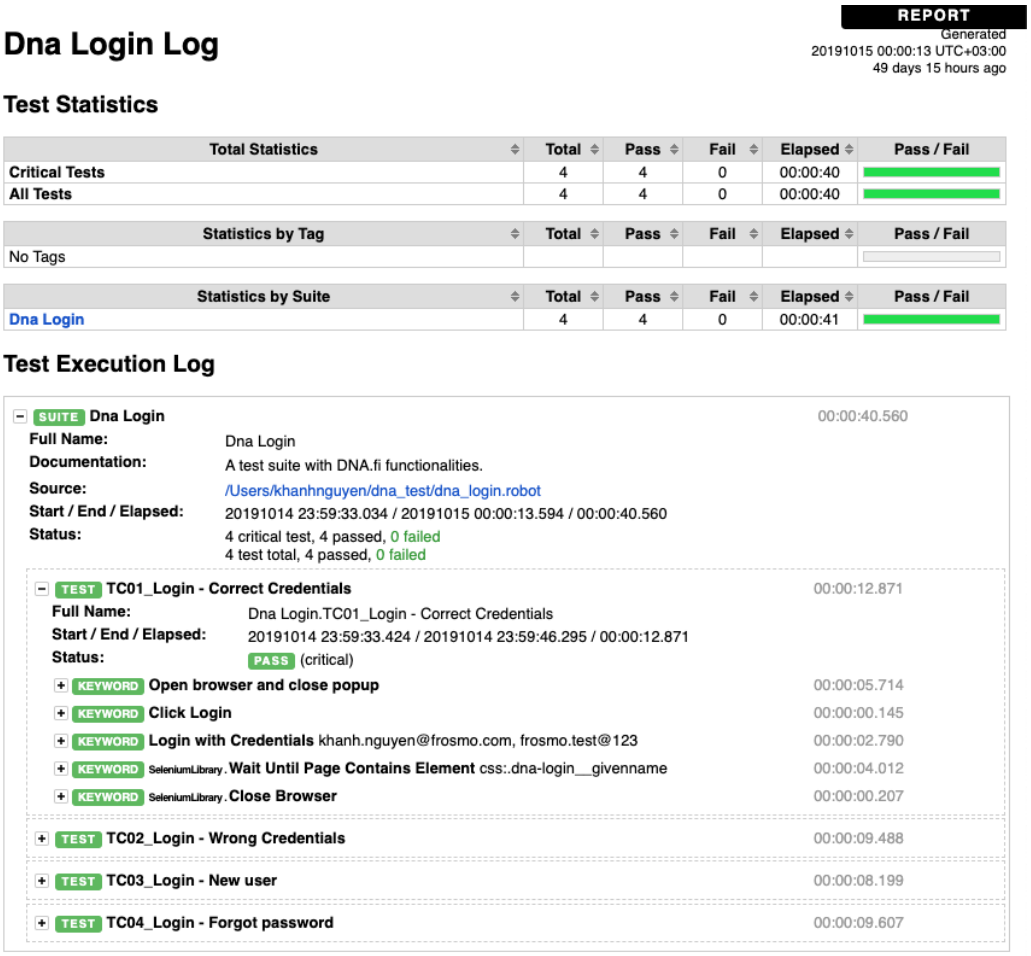
The second problem was that tests are very fragile. This is partly due to SeleniumLibrary: such a trivial task as a button click executed inconsistently. Implicit and explicit wait need to be included for test to succeed. There are some issues with performance where browser becomes more sluggish the longer the test session. For that reason, some test cases, which succeed when executed alone, fail in test suite execution. Another factor is the dynamic nature of client website. A product could be out of stock or a phone package could be taken off the store in a daily basis.

Security of storing AWS token

Another problem was AWS credential needed for pipeline execution. GitLab CI provides an easy way to store these keys in environment variables in

pipeline settings of the repositories. But if they are stored in the same GitLab project where the test code and CI file are located, they will be available for all users who have access to that repository. This is not a problem in the pilot case and it was also implemented in a separated repository. However, some projects enforce strict permission of access to production. In order to limit the access, separated repository need to be created for test. This could increase the complexity of the pipeline.

The results of the implemented pipeline evaluation are described in chapter 6.



Dna Login Log

REPORT
Generated
20191015 00:00:13 UTC+03:00
49 days 15 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	4	4	0	00:00:40	
All Tests	4	4	0	00:00:40	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Dna Login	4	4	0	00:00:41	

Test Execution Log

SUITE Dna Login 00:00:40.560

Full Name: Dna Login
Documentation: A test suite with DNA.fi functionalities.
Source: /Users/khanhnguyen/dna_test/dna_login.robot
Start / End / Elapsed: 20191014 23:59:33.034 / 20191015 00:00:13.594 / 00:00:40.560
Status: 4 critical test, 4 passed, 0 failed
4 test total, 4 passed, 0 failed

TEST TC01_Login - Correct Credentials 00:00:12.871

Full Name: Dna Login.TC01_Login - Correct Credentials
Start / End / Elapsed: 20191014 23:59:33.424 / 20191014 23:59:46.295 / 00:00:12.871
Status: PASS (critical)

KEYWORD Open browser and close popup 00:00:05.714
KEYWORD Click Login 00:00:00.145
KEYWORD Login with Credentials khanh.nguyen@frosmo.com, frosmo.test@123 00:00:02.790
KEYWORD SeleniumLibrary . Wait Until Page Contains Element css:.dna-login__givenname 00:00:04.012
KEYWORD SeleniumLibrary . Close Browser 00:00:00.207

TEST TC02_Login - Wrong Credentials 00:00:09.488

TEST TC03_Login - New user 00:00:08.199

TEST TC04_Login - Forgot password 00:00:09.607

Figure 5.4: An example test report

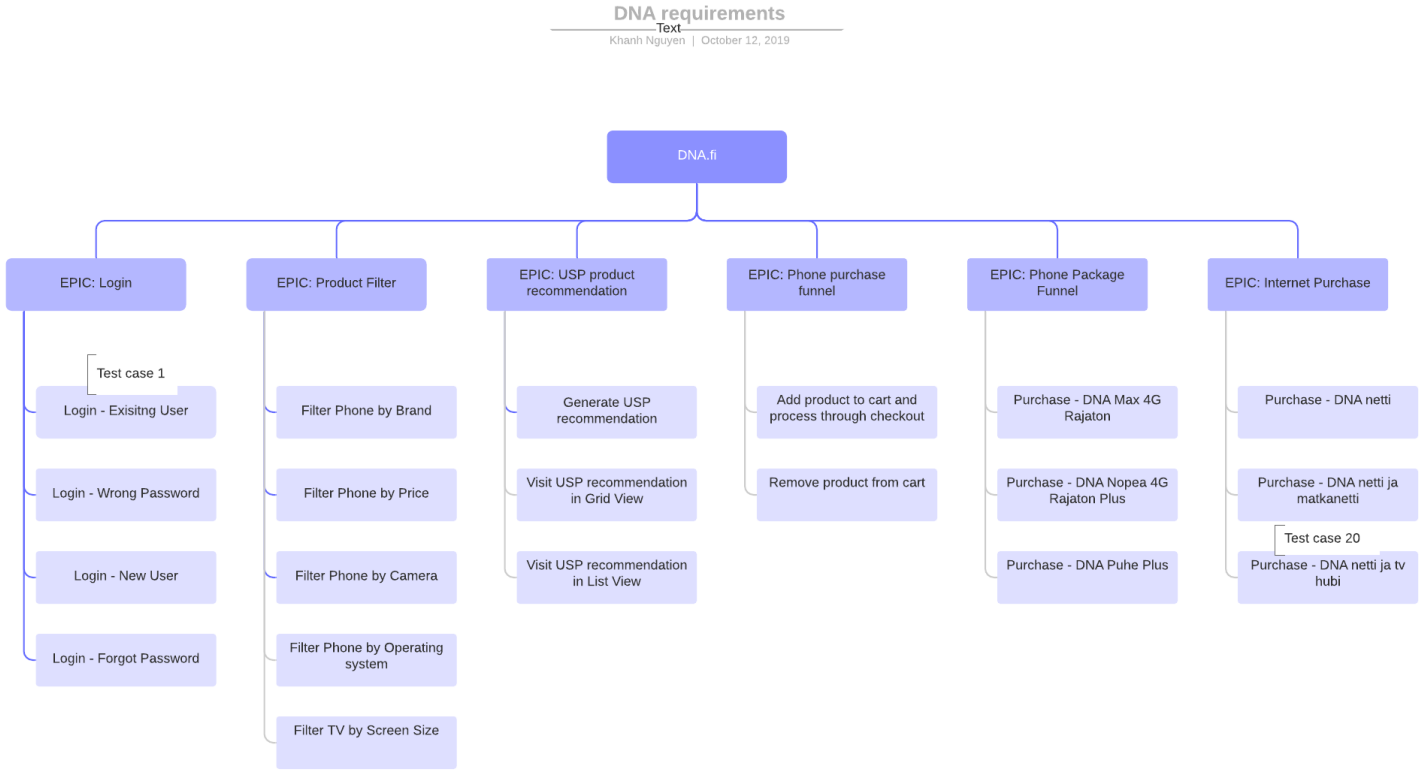


Figure 5.7: DNA.fi test cases

Chapter 6

Evaluation

This chapter elaborates the evaluation of the implemented pipeline. The first section discusses what features have been implemented in accordance with proposed requirements in Chapter 4. The next section evaluates the test automation system. The last section summarized feedback from stakeholders.

6.1 Implemented end-to-end pipeline

The final realization of requirements in Chapter 4 were presented in Table 6.1.

6.1.1 Test Programming

Use GitLab as Version Control System to store E2E test code

GitLab is used as Version Control System for E2E test code. GitLab CI was used for the whole CI/CD pipeline.

Separated repositories for pilot projects

Pilot cases were developed in a separated repository. Requirements stated that test code will be merged with client project repositories after validation and refinement. However, there was a later realization of some projects where access key permission should be restricted in GitLab CI. It was decided that test repository will be kept in a distinct repository.

Requirements	**Category**	**Importance**	**Implemented**
Use GitLab as VCS	Programming	10	Yes
Setup test environment with AWS	Running	9	Yes
Test data storage in Amazon S3	Monitoring	8	Yes
Run test on schedule	Running	8	Yes
Run test on different type of browsers	Running	7	Yes
HTML reports	Monitoring	7	Yes
Separated repositories for pilot projects	Programming	6	Yes
Alert via Slack	Monitoring	6	Yes
Use Docker as test container	Running	5	Yes
Branching and merge request for peer review	Programming	5	Yes
Same repository as client project for E2E test	Programming	3	No
Run test automatically on commit	Running	3	Yes
Alert via Kibana integration	Monitoring	2	No
Test data storage in FCP	Monitoring	1	No
Code coverage is measured	Monitoring	1	No
Graphical statistic	Monitoring	1	No
Alert via email	Monitoring	0	Yes

Table 6.1: Completion of Requirements

Branching and merge request for peer review / Same repository as client project for E2E test

In the pilot case, there was only one developer and branching and merge request for peer review have not been employed. However, branching and merge request come natively with GitLab setup. This process can be enable later in projects if needed.

6.1.2 Test Running

Run tests on different types of browser

The main motive of choosing Robot Framework is because this framework supports a wide range of browsers. *SeleniumLibrary* in Robot Framework supports all WebDriver implementation names: Firefox, Chrome, IE, Opera, Safari, PhantomJS or Remote. Simply by changing execution parameter, test code will be runnable in all browsers listed above.

In the pilot case, Docker image with pre-installed Chrome and Firefox was used.

Run tests automatically on commit / Run test on schedule

GitLab CI was setup to run on commit on master branch. There is also a schedule setup to run the pipeline at 4 am everyday. Previously, it was set at midnight but 4 am was chosen later since it was the least busy time of the day.

Use Docker as test container

The E2E test run in a Docker container with required environment for Robot Framework. Docker image *ppodgorsek/robot-framework* [1] was used. This is a Docker Image based on Docker Alpine with Robot Framework, SeleniumLibrary, Firefox and Chrome.

[1] https://github.com/ppodgorsek/docker-robot-framework

6.1.3 Monitoring

HTML reports / Test data storage in Amazon S3

The reports and logs from each run are saved as artifacts in GitLab. These artifacts are available for downloading within a 4-week period. Reports and logs are presented in HTML format and copied to Amazon S3 bucket for long-term storage.

Alert via Slack & email

Slack provide Webhook URL, which can be used to send automated messages to Slack channel. GitLab Setting has an integration option for Slack notification. Notifications on failed jobs are sent directly to an internal Slack channel. GitLab integration also provides Pipeline emails, which notify a list of recipients on the events of broken pipelines.

6.2 Evaluation of the test automation system

6.2.1 Tests

This section evaluates the quality of the tests: extendibility, maintainability and reliability of the tests.

The cost for maintenance and development largely depend on how the tests are developed. For the time being, tests are written in Robot Framework SeleniumLibrary. Keywords are used extensively and the test cases are designed to be small and reusable. Most of the alterations needed, when creating a new test, are usually DOM element selectors. This establishes a good basis for the extension of test coverage in the future. However, it is important to constantly pay attention to reusability and maintainability while developing in order to keep the code quality from deteriorate. It is also noted that E2E tests are expensive and less robust that other level of testing. Therefore, E2E tests should focus on regression type of testing. Existing features, main functionalities that are changed less often should be the point of interest when developing new tests.

The reliability of the tests has varied. Tests can be seen as relatively unreliable. Most failure cases happened when the client website changed. Usually these changes do not break Frosmo feature. Nevertheless, the developers feel the swift notifications are convenient. They do not need to rely on clients to notify of the changes. On the other hand, too many ”soft alerts” may cause developers to lose attention. They eventually ignore the notice

or abandon the tests. In addition, there are some false positive cases such as insufficient wait-time for clicking and loading web page. The causes for these errors are limited and similar. They could all be identified and test reliability will improve overtime.

In conclusion, this test system will be useful for regression testing. The test architecture enables cost effective development and maintenance. Reliability of the test is not very good at the moment, but it could be improved.

6.2.2 Test execution and report

This section discusses the usefulness of test execution and reporting for developers.

The pilot case can be introduced to other clients with minimum efforts. Test repositories can be duplicated with very few alterations needed (e.g. test report script, files naming). Almost all defined keywords are reusable. The test schedule can be easily configured in GitLab CI. The test executed itself inside a Docker container, which eliminates the need for setting-up and configuring test environment.

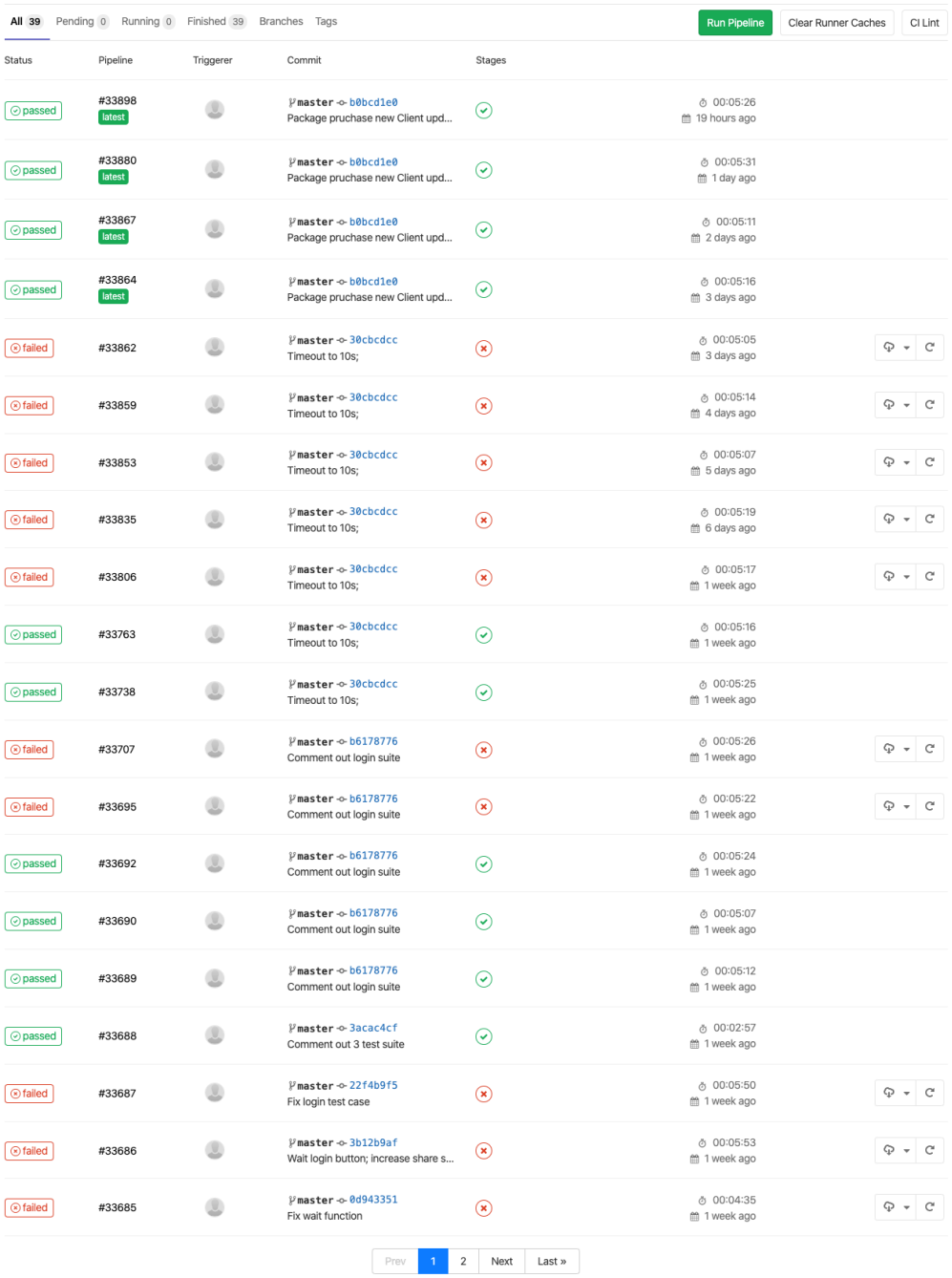


Figure 6.1: GitLab pipeline view

Test execution time varied from 5 minutes 7 seconds to 5 minutes 31 seconds. The test suite includes 16 test cases with about 250 steps. The execution time could be improved by not having unnecessary long waiting times between various steps. Tests are run on GitLab server, which is hosted by company. The only external infrastructure cost is AWS S3 storage.

The test reports are sufficient. Error messages are listed clearly with screen shots. One problem is that a single error produces too many reports. For example, an out-of-stock product, could cause all of the purchase funnel tests to fail, because adding a product to the cart is the first part of all the tests. There would need to be updates in test cases or validations that prevent the same error from being reported multiple time.

In the meantime, test reports are streamed directly to developers via Slack channel and emails. However, if test reliability becomes a big problem then developers may ignore the alerts. A solution could be to use automated tests as helper in finding defects. Found defects must then be manually verified by QA before posting Jira tickets. This ensures that the reports are accurate and shield developers from noise. However, the manual process means that it would take days before the defects are reported if the test execution is not monitored closely.

Despite all the problems above, current test execution and reporting system are still helpful in providing confidence with newly developed features. Furthermore, with the base infrastructure in place to automatically execute and report tests, it would be easier to improve the details and automation level later when the tests are more robust.

6.3 Feedback from stakeholders

This section summarized results from the final review workshop. The ideas and comments are categorized by related requirements.

6.3.1 Test Programming

"It feels not right having to manage two separated repositories"

The first matter was that pilot cases were developed in a separated repository. It was decided that, in some project, test repository are kept in a distinct repository to restrict access to other AWS key. The team raised this issue, worrying that this may over-complicate the existing process. Moreover, broken pipeline in E2E test repository would not stop the main repository from deploying. However, this is only the case for a few projects. GitLab

CI/CD also support multi-project pipelines [2]. Multi-project pipelines are not foreign to large products that require cross-project dependencies, especially those that adopting a micro-services architecture.

In general, the both repository schemes do work well and could be selected according to the project.

6.3.2 Test Running

"Who will be in charge of these CI config ?"

The team all agree on GitLab being a powerful CI/CD tool and Docker image really simplify E2E test setup. One important feedback was that now developers and QAs will be involved with DevOps tasks, which require additional skills such as GitLab CI config, Docker and bash script. The company culture does encourage people to gain knowledge and DevOps will become more and more imporant as an expertise. All team members should take initiative in writing and support their project CI/CD.

"It may be okay for now. But what happen if the cost or execution time of the test grow too much ?"

The workshop participants noted that the execution price of the E2E test are very low at the moment. But the team should pay attention on the execution time and memory load to avoid browser crashes, especially in the future when test case grow bigger. In the event of test execution becomes too expensive, it could be moved onto AWS. Other compromise might be to run intensive E2E tests on feature release and/or keep the number of E2E tests at minimum. However, with current setup, the probability of high cost is very rare.

6.3.3 Monitoring

"There are too much notification via Slack channel and emails"

There are some feedback about spamming message on Slack and emails. This was due to some misconfiguration in the beginning where each failed case will trigger one message. A fix was introduced later to prevent an early failure that may break the whole test suite. The amount of message is now strictly one per failed pipeline job. Team can also configure the frequency based on their need.

[2] https://docs.gitlab.com/ee/ci/multi_project_pipelines.html

Chapter 7

Discussion

7.1 RQ1: What are the requirements for the end-to-end testing pipeline in the case project?

The first RQ was about gathering all the requirements for the E2E testing pipeline that could be applied to the case project, and then be used in all other company clients. Motivated by the points made in section 1.1, the E2E testing will focus on regression testing when client or Frosmo made changes to the website. The require elicitation process was done in three steps: interviews with the stakeholders, brainstorming session among developers, workshop to prioritize requirements. The result of the requirements selected is presented in Table 4.1. Some of the key findings of requirements elicitation are:

- Current company project setup has some requirement about the development process and tools. CI tool and repository manager will be GitLab. Docker and AWS will be used to run test. Amazon S3 will be used to store test reports.
- The pilot project will have its test repository separated from the main repository. After validation and refinement,tests can be moved into client project repository.
- E2E tests should be able to run on multiple browsers. Client projects are mostly e-commerce websites who prioritize user reach and accessibility. Therefore, compatibility testing has always been important. As a result, the new automation E2E testing should be able to cover testing on multiple browsers.

- Test reports should be available in HTML or graphical format. Issues found should be notified on email and Slack channel.

The result requirements correspond to the findings in existing studies about implementation of E2E testing [32, 33, 34]. Later implementation also did not reveal contradiction to CI/CD and monitoring practices.

7.2 RQ2: How to implement End-to-end testing in WEMP?

After studying existing E2E testing tools and practices, considering the requirements of the project, Robot framework was selected as the framework to implement E2E testing. The list of studied tools were shown in Table 2.2. Robot framework were selected due to a number of advantages: high customizability, modularity, wide range of support with libraries and tools, easy to use tabular syntax and compatibility with all popular browsers (thanks to SeleniumLibrary [1]).

The project architecture is presented in Figure 5.5. The test GitLab repository will store Robot framework tests. Code commit into this repository will trigger the E2E testing pipeline. First, GitLab CI/CD will start the GitLab runner that spin up a Docker-in-Docker container. This container will then run the Robot framework testing container. The test will be executed inside this container. Browsers will be opened and run the tests on the website as programmed. After the tests complete, test report will be copied to Amazon S3 to store. Any failure will be notified via email and Slack channel.

E2E tests should act as a regression test, which means it should be run upon release of Frosmo feature onto client websites. The test should also be scheduled to run everyday to verify system health, since client development pipelines are out of Frosmo scope. With that in mind, the tests should be kept at minimum amount, covering only crucial customer journey: login, purchase funnel, Frosmo features, etc.

7.3 RQ3: Does end-to-end testing implementation meets the requirements in RQ1?

The realization of requirements was presented in Table 6.1. All requirements of importance above 5 were implemented. Except for the requirement of

[1] https://robotframework.org/Selenium2Library/Selenium2Library.html

same repository, which were later ruled as not possible, all requirements of importance above 2 were implemented. The separated repository, however, create a good base for future cases. It is easier to clone the test repository and setup new project. The separation of repository does not hinder CI job execution either, since GitLab CI/CD support multi-repository pipeline.

The prior research showed that E2E testing is costly and easy to break since they drive through UI and touch every component in the system. Implementation also reveals that E2E test are very fragile, fail after every few days of running. After a number of modifications and optimizations, tests are currently in a more stable state. On the other hand, during the time of testing, some of the change in client website that may break Frosmo features were detected swiftly.

Test monitoring implementation meets all requirements. Issue notification via email and Slack are scalable and configurable according to project need. Test reports stored in Amazon S3 also working as expected. There is no initial requirement about the cost of pipeline execution, but estimated expenditure was calculated as very low, since most of the jobs were running on GitLab server which located in the company premise. The price of production and release execution was not calculated because it is not related to the testing pipeline and mostly depends on the setup of each project. However, the current setup is highly customizable and it can be modified to fit the requirement of other client cases.

7.4 Limitation

During the requirement elicitation process, there were no clients included in the interview despite playing an important part in the project. Stakeholders were divided into three group: QA, Project manager and developers . This is due to the nature of the project is highly experimental and most communication with clients is done remotely making it hard to include them into the process.

On the implementation aspect, test run period was identified as relatively short: about 2 months. During this time frame, there was no real incident where the pipeline can prove it true necessity and usability.

Chapter 8

Conclusions

The goal of this thesis is to implement an E2E testing into the existing pipeline of an Web Experience Management Platform. The process was based on the design science research methodology with a series of workshops and technical implementation. Project requirements were gathered through an elicitation procedure include interviews with stakeholders. The literature review was conducted to gain an understanding about the current state of E2E testing, tools and technology. Based on the review, Robot Framework was selected as the E2E testing framework. GitLab was used as CI/CD solution. The final evaluation workshop showed that all high priority requirements for the pipeline were completed.

The research showing that, despite being the best way to verify the SUT, E2E test are very expensive. Automating E2E test is one answer to this, but it is challenging because E2E tests tend to break often. They are expensive to develop, execute and yet also lack robustness.

Reality of the project conform with the existing research. E2E tests are often breaking or giving inconsistent results. A fair amount of development time was dedicated to strengthen the logic and prepare for exceptions during test run. As suggested by literature, tests were limited to the most crucial customer journey to reduce the amount of test cases.

It can be seen that the result of this study contributes to the research in the field. In spite of the large amount of non-academic materials available in web article or blog posts, this topic is not well presented in academic papers. On the other hand, the achieved result cannot be generalized because they were conducted for a single case project with company-specific requirements. However, the common aspects of the pipeline with E2E automated tests will be reused for other client in the case company, which could be seen as positive.

There are rooms for improvement on the project as future work. For

maintainability and scalability, the E2E test code should be implemented in the same repositories with the current code base. Another improvement would be to introduce customer personnels into the project to gather more meaningful requirements and feedbacks. In future development for other clients, graphical statistic or integration with existing logging system would be beneficial as well.

Bibliography

[1] S. Sampath and S. Sprenkle, "Chapter Four - Advances in Web Application Testing, 2010–2014," in *Advances in Computers*. Elsevier, 2016, vol. 101, pp. 155–191.

[2] D. R. Lakshmi and S. S. Mallika, "A Review on Web Application Testing and its Current Research Directions," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 4, p. 2132, Aug. 2017.

[3] V. Debroy, L. Brimble, M. Yost, and A. Erry, "Automating Web Application Testing from the Ground Up: Experiences and Lessons Learned in an Industrial Setting," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. Vasteras: IEEE, Apr. 2018, pp. 354–362.

[4] R. Rwemalika, M. Kintis, M. Papadakis, and Y. L. Traon, "Can we automate away the main challenges of end-to-end testing?" *11-Dec-2018*, p. 5, 2018.

[5] E. Alégroth, R. Feldt, and L. Ryrholm, "Visual GUI testing in practice: Challenges, problemsand limitations," *Empirical Software Engineering*, vol. 20, no. 3, pp. 694–744, Jun. 2015.

[6] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, K. Petersen, and M. V. Mantyla, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *2012 7th International Workshop on Automation of Software Test (AST)*. Zurich, Switzerland: IEEE, Jun. 2012, pp. 36–42.

[7] B. García, F. Gortázar, M. Gallego, and E. Jiménez, "User Impersonation as a Service in End-to-End Testing:," in *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 707–714.

[8] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*, ser. The Addison-Wesley Signature Series. Upper Saddle River, NJ: Addison-Wesley, 2010.

[9] "What is an End-to-End Test? - Definition from Techopedia." [Online]. Available: http://www.techopedia.com/definition/7035/end-to-end-test

[10] "End-to-end Testing — GitLab." [Online]. Available: https://docs.gitlab.com/ee/development/testing_guide/end_to_end/index.html

[11] R. Black, E. van Veenendaal, and D. Graham, *Foundations of Software Testing: ISTQB Certification*, 3rd ed. Andover: Cengage Learning, 2012.

[12] B. Garcia, *Mastering Software Testing with JUnit 5: Comprehensive Guide to Develop High Quality Java Applications*, 2017.

[13] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," 2001. [Online]. Available: http://www.agilemanifesto.org/

[14] J. Gregory and L. Crispin, *More Agile Testing: Learning Journeys for the Whole Team*, ser. Addison-Wesley Signature Series. Upper Saddle River, NJ: Addison-Wesley, 2015.

[15] M. Cohn, "The Forgotten Layer of the Test Automation Pyramid." [Online]. Available: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

[16] "Testing levels — GitLab." [Online]. Available: https://docs.gitlab.com/ee/development/testing_guide/testing_levels.html

[17] "The Practical Test Pyramid." [Online]. Available: https://martinfowler.com/articles/practical-test-pyramid.html

[18] "How to Perform End-to-End Testing." [Online]. Available: https://smartbear.com/learn/automated-testing/how-to-perform-end-to-end-testing/

[19] Martin Fowler, "Test Pyramid." [Online]. Available: https://martinfowler.com/bliki/TestPyramid.html

[20] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Chapter Five - Approaches and Tools for Automated End-to-End Web Testing," in *Advances in Computers*, A. Memon, Ed. Elsevier, 2016, vol. 101, pp. 193–237.

[21] S. R. Choudhary, D. Zhao, H. Versee, and A. Orso, "WATER: Web Application TEst Repair," in *Proceedings of the First International Workshop on End-to-End Test Script Engineering - ETSE '11*. Toronto, Ontario, Canada: ACM Press, 2011, pp. 24–29.

[22] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis - ISSTA 2015*. Baltimore, MD, USA: ACM Press, 2015, pp. 385–396.

[23] S. H. Jensen, S. Thummalapenta, S. Sinha, and S. Chandra, "Test Generation from Business Rules," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. Graz, Austria: IEEE, Apr. 2015, pp. 1–10.

[24] T. Yue, S. Ali, and M. Zhang, "RTCM: A natural language based, automated, and practical test case generation framework," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis - ISSTA 2015*. Baltimore, MD, USA: ACM Press, 2015, pp. 397–408.

[25] I. L. Araújo, I. S. Santos, J. B. F. Filho, R. M. C. Andrade, and P. S. Neto, "Generating test cases and procedures from use cases in dynamic software product lines," in *Proceedings of the Symposium on Applied Computing - SAC '17*. Marrakech, Morocco: ACM Press, 2017, pp. 1296–1301.

[26] B. Lima and J. P. Faria, "A Survey on Testing Distributed and Heterogeneous Systems: The State of the Practice," in *Software Technologies*, E. Cabello, J. Cardoso, A. Ludwig, L. A. Maciaszek, and M. van Sinderen, Eds. Cham: Springer International Publishing, 2017, vol. 743, pp. 88–107.

[27] "Examples of Test Oracles." [Online]. Available: http://www.testingeducation.org/k04/OracleExamples.htm

[28] S. Casteleyn, G. Rossi, M. Winckler, and ICWE, Eds., *Visual vs. DOM-Based Web Locators: An Empirical Study (IN Web Engineering: 14th*

International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014 ; Proceedings), ser. Lecture Notes in Computer Science Information Systems and Application, Incl. Internet/Web and HCI. Cham: Springer, 2014, no. 8541, oCLC: 896700386.

[29] "SikuliX," 2019. [Online]. Available: https://github.com/RaiMan/SikuliX1

[30] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, Jan. 2017.

[31] ——, "Continuous software engineering and beyond: Trends and challenges," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering - RCoSE 2014*. Hyderabad, India: ACM Press, 2014, pp. 1–9.

[32] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Upper Saddle River, NJ: Addison-Wesley, 2010.

[33] L. Bass, I. M. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, ser. The SEI Series in Software Engineering. New York: Addison-Wesley Professional, 2015.

[34] G. Kim, P. Debois, J. Willis, J. Humble, and J. Allspaw, *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*, first edition ed. Portland, OR: IT Revolution Press, LLC, 2016, oCLC: ocn907166314.

[35] "Eggplant Functional." [Online]. Available: https://eggplant.io/products/dai/eggplant-functional

[36] "Maveryx." [Online]. Available: https://www.maveryx.com

[37] A. R. Hevner and S. Chatterjee, *Design Research in Information Systems: Theory and Practice*, ser. Integrated Series in Information Systems. New York ; London: Springer, 2010, no. v. 22.

[38] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, Dec. 2007.

[39] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering - ICSE '00*. Limerick, Ireland: ACM Press, 2000, pp. 35–46.

[40] "Kibana." [Online]. Available: https://www.elastic.co/products/kibana

[41] "RIDE - Test data editor for Robot Framework." [Online]. Available: https://github.com/robotframework/RIDE

[42] "Visual Studio Code." [Online]. Available: https://code.visualstudio.com/

[43] "Docker Hub." [Online]. Available: https://hub.docker.com/

[44] D. Ståhl and J. Bosch, "Automated software integration flows in industry: A multiple-case study," in *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*. Hyderabad, India: ACM Press, 2014, pp. 54–63.